Lecture 5

Python Basics

Learning Objectives Today

- Python Variables
- Data Types
- String
- Casting
- List
- Tuple
- Dictionary
- Operators

Python Variables

- age = 30
- name = "Alice"
- pi = 3.14159

- Variables are often described as boxes you can store values in.

Variable Features

- Redeclaration
- Multiple Assignment

```
num1 = 5
num1 = 3.45

num1, num2, stri = 4, 5, "okay"
num1 = num2 = num3 = 10

print(num1, num2, num3)
```

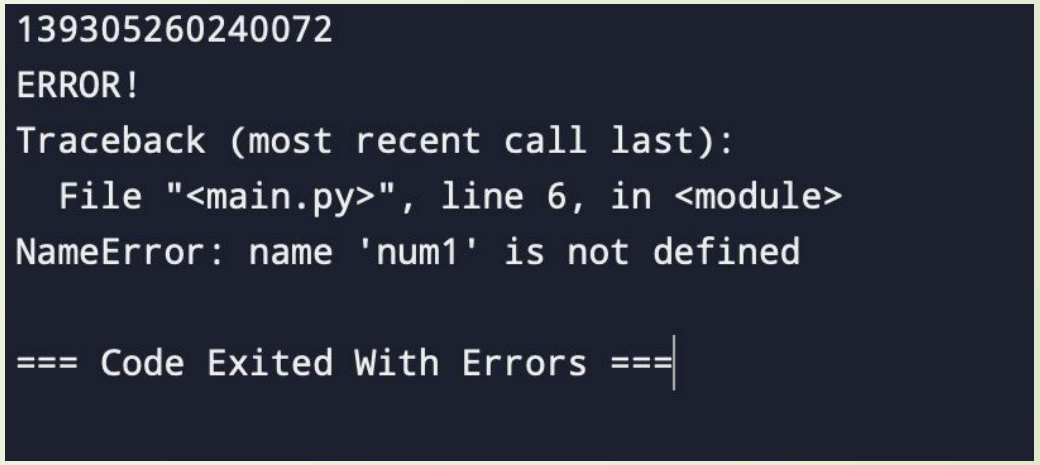
Delete Variable

Num1 = 10

print(id(num1)) #Shows the RAM location where this is saved

del num1

print(num1)



```
139305260240072
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 6, in <module>
NameError: name 'num1' is not defined

=== Code Exited With Errors ===
```

Python Data Types

Numeric Types:

Integers (int): Represent whole numbers, positive or negative (e.g., age = 30, score = -10).

Floats (float): Represent decimal numbers (e.g., pi = 3.14159, temperature = 25.5).

Complex Numbers (complex): (e.g., z = 3+5j).

Text Types:

Strings (str): Represent sequences of characters, enclosed in single or double quotes (e.g., name = "Alice", message = 'Hello, world!').

Sequence Types:

Lists (list): Ordered, mutable collections of items (e.g., fruits = ["apple", "banana", 10]).

Tuples (tuple): Ordered, immutable collections of items (e.g., coordinates = (10, 20))

Set Types:

Sets (set): Unordered collections of unique items enclosed in curly braces {}. Useful for checking membership and removing duplicates (e.g., unique_numbers = {1, 2, 2, 3, 4}).

Frozen Sets (frozenset): Immutable versions of sets (e.g., frozen_numbers = frozenset({1, 2, 3})).

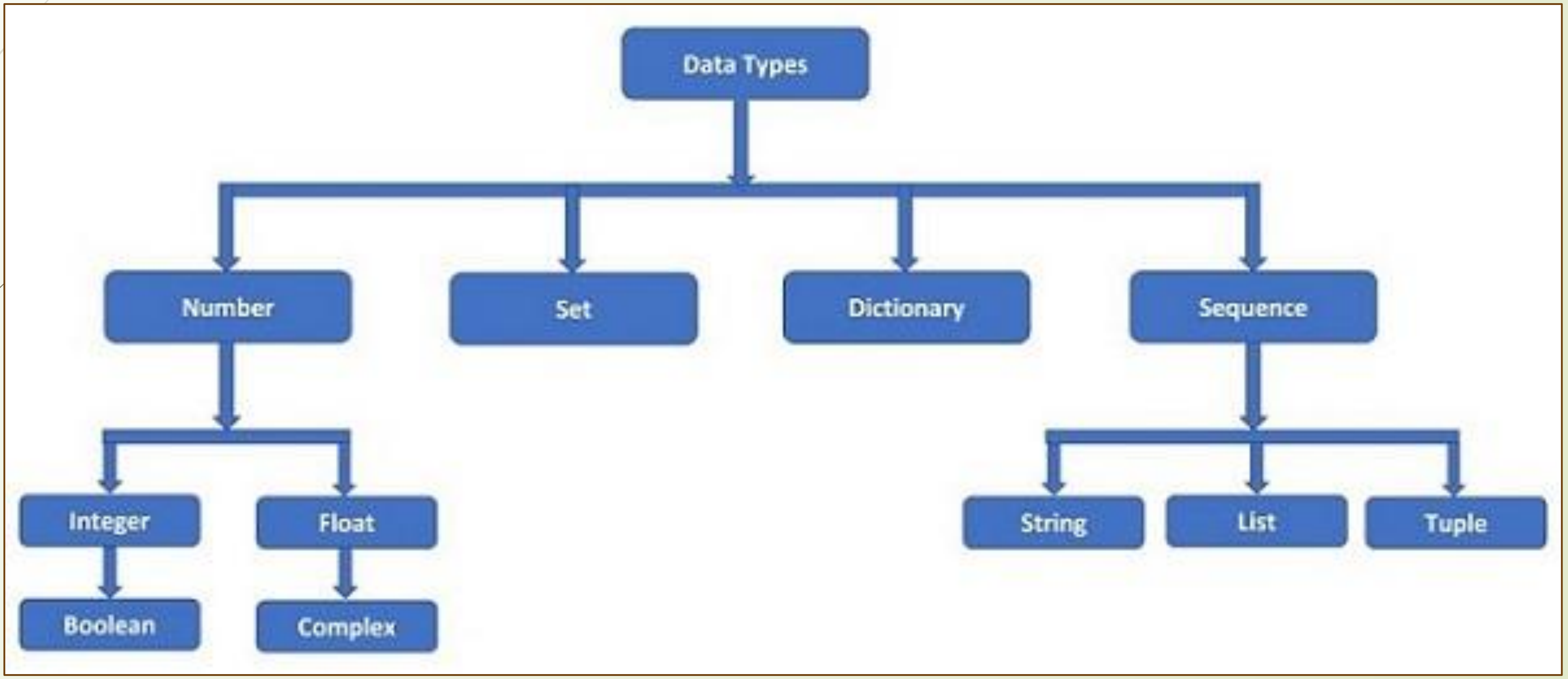
Mapping Types:

Dictionaries (dict): Unordered collections of key-value pairs enclosed in curly braces {}. Keys must be unique and immutable (e.g., person = {"name": "Bob", "age": 30}).

Other Types:

Boolean (bool): Represent logical values, True or False (e.g., is_running = True).

Data Types


Data Types
Number
Set
Dictionary
Sequence
Integer
Float
Boolean
Complex
String
List
Tuple

Try Out

- Input a number and print
- Input a number 1451.1545 into the variable and output only two decimal values
- Take two strings and an integer. Print a single string.
- Multiple Variable Assignment
- Delete a variable

Casting

```
x = int(1)
y = int(2.8)
z = int("3")
print(x)
print(y)
print(z)
```

```
x = float(1)      # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2
print(x)
print(y)
print(z)
print(w)
```

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
print(x)
print(y)
print(z)
```

Implicit Type Casting (Automatic Conversion)
Explicit Type Casting (Manual)

String Formatting and Functions

```
message = 'Hello, world!'  # Single quotes
name = "Alice"  # Double quotes
multiline_text = """This is a string
that spans multiple lines."""  # Triple quotes

print(multiline_text)
```

String

Accessing Characters

```
first_letter = message[0]  # 'H'
last_letter = message[-1]  # '!'
print(first_letter, last_letter)
```

Slicing

Extract substrings using colon (:) notation within square brackets.

Syntax: [start:end:step].

```
message = "PythonClass"
substring = message[1:6]  # "world" (extracts characters from index 6 to 10)
print(substring)
substring = message[1:8:2]  # "world" (extracts characters from index 6 to 10)
print(substring)
```

Try Out

- A Dataset contains strings in a column. The last character of the sentence contains the currency type. And 5th,6th, and 7th letters contain the amount of money. Can you extract it?

String

- Strings provide various built-in methods for manipulation:
- upper(): Convert to uppercase (e.g., message.upper()).
- lower(): Convert to lowercase (e.g., message.lower()).
- strip(): Remove leading and trailing whitespaces (e.g., message.strip()).
- find(substring): Find the index of the first occurrence of a substring (returns -1 if not found).
- replace(old, new): Replace all occurrences of a substring with another substring.
- Many more! (Refer to Python documentation for the full list)

Try Out

- Apply the functions and see what happens to the string.

String Formatting

```
name = "Bob"
age = 30
formatted_text = f"Hello, {name}! You are {age} years old."
print(formatted_text)  # Output: Hello, Bob! You are 30 years old.
```

Try OUT

String Space Removal

```
message = "   Hello, World!    "

# Remove all leading/trailing whitespaces
stripped_message = message.strip()
print(stripped_message)  # Output: Hello, World!

# Remove leading whitespaces
left_stripped_message = message.lstrip()
print(left_stripped_message)  # Output:Hello, World!

# Remove trailing whitespaces
right_stripped_message = message.rstrip()
print(right_stripped_message)  #    Hello, World!
```

String Search

```
message = "Hello, brave new world!"

# Find the index of the first occurrence of "world"
world_index = message.find("world")
print(world_index)  # Output: 7

# Find the index of the last occurrence of "world" (if it appears multiple times)
last_world_index = message.rfind("world")
print(last_world_index)  # Output: 7
```

String Split and Join

```
message = "Hello, how are you today?"

# Split the string into words using spaces as separator
words = message.split(" ")
print(words)  # Output: ['Hello,', 'how', 'are', 'you', 'today?']

message2 = "I am Atanu $ I'm taking your class"
words = message2.split("$")
print(words)  # Output: ['Hello,', 'how', 'are', 'you', 'today?']

# Join the words back into a string with underscores
joined_message = "_".join(words)
print(joined_message)  # Output: Hello,_how,_are,_you,_today?
```

String Replacement

```
message = "Hi, this is a test string."

# Replace all occurrences of "is" with "was"
replaced_message = message.replace("is", "was")
print(replaced_message)  # Output: Hi, this was a test string.

# Replace only the first occurrence
replaced_message = message.replace("is", "was", 1)
print(replaced_message)  # Output: Hi, this was a test string.
```

Try Out

- You are given a list of strings representing words. Your task is to write a Python program that performs the following operations:
- Join all the words in the list into a single string, separated by a specific delimiter.
- Split the resulting string back into individual words using the same delimiter.
- Write a Python function join_split_words(words_list, delimiter) that takes a list of words and a delimiter as input and returns the list of words obtained after joining and splitting the original list.
  - Input:
    - words_list: A list of strings representing words (e.g., ["Hello", "World", "Python"])
    - delimiter: A string representing the delimiter to use for joining and splitting (e.g., ",")
  - Output:
    - A list of strings obtained after joining and splitting the original list using the specified delimiter.

Try Out

- Example

```
words_list = ["Hello", "World", "Python"]
delimiter = ","
result = join_split_words(words_list, delimiter)
print(result)
```

Expected Output:

```
['Hello', 'World', 'Python’]
```

Lets t

okay lets try formating Hello, can you see World ry formatted output????????????

Lists vs Array in Python

- The main differences are:
  - Arrays can only store elements of the same data type, while lists can store mixed data types.
  - Arrays are more memory-efficient and faster for numerical operations, while lists consume more memory but are more flexible.
  - Arrays require explicitly importing the array module, while lists are a built-in data structure in Python.
  - Arrays have limited functionality compared to the extensive list of built-in methods available for lists.
  - Modifying the size of an array is more difficult than modifying a list, which can be done dynamically.

Lists vs Array in Python

- **Choosing Between Lists and Arrays:**
  - **General purpose collection:** Use lists for general-purpose data storage where you might have different data types and need flexibility in adding, removing, or modifying elements.
  - **Numerical computations:** If you're working with large amounts of numerical data and need efficient operations, consider using arrays from NumPy. NumPy arrays offer a powerful set of functions for numerical computations and vectorized operations.
- **Additional Considerations:**
  - For simple numerical arrays, the array module can be an option, but NumPy provides a richer set of features and optimizations.
  - If you don't need the performance benefits of arrays for numerical data, lists are generally more convenient due to their built-in nature and flexibility.

List

- List is one of the built-in data types in Python. A Python list is a sequence of comma separated items, enclosed in square brackets [ ]. The items in a Python list need not be of the same data type.

```
list1 = ["Rohan", "Physics", 21, 69.75]
list2 = [1, 2, 3, 4, 5]
list3 = ["a", "b", "c", "d"]
list4 = [25.50, True, -55, 1+2j]
```

List

```
#Access
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7 ];
print ("list1[0]: ", list1[0])
print ("list2[1:5]: ", list2[1:5])

#Update
list = ['physics', 'chemistry', 1997, 2000];
print ("Value available at index 2 : ")
print (list[2])
list[2] = 2001;
print ("New value available at index 2 : ")
print (list[2])
```

List

```
#Delete
list1 = ['physics', 'chemistry', 1997, 2000];
print (list1)
del list1[2];
print ("After deleting value at index 2 : ")
print (list1)
```

Negative Access??

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized’]
print(bicycles[-1])
```

List

- Insert vs Append

```
bicycles.append("Duranta")
print(bicycles)

bicycles.insert(3, "Racer")
print(bicycles)
```

Try Out

- Insert, Append in a Python List
- Delete values from a list
- Search and delete a value from a Python list
- **Search and delete all occurances of a value from the list**

Solutions are in the python scripts of lecture 5

List

- **Slice and PoP**

```
fruits = ["apple", "banana", "cherry"]
sublist = fruits[1:3]  # ["banana", "cherry"] (extracts elements at index 1 and 2, excluding index 3)

print(sublist)

removed_fruit = fruits.pop(2)  # removed_fruit is "cherry" and fruits becomes ["apple", "orange", "mango"]
print(removed_fruit)
```

List Functions

- **Sort**

```
my_list = [66, 33, 22, 44, 88, 77, 99, 11]
my_list2 = sorted(my_list)
print(my_list2)

my_list.sort()
print(my_list)
```

List Functions

- **Sort**

```
numbers = [3, 1, 4, 2]
numbers.sort()
print(numbers)  # Output: [1, 2, 3, 4]

numbers.sort(reverse=True)
print(numbers)  # Output: [4, 3, 2, 1]

# Sorting by length of strings
strings = ["aa", "bbbbb", "ccc"]
strings.sort(key=len)
print(strings)  # Output: ["aa", "ccc", "bbbbb"]
```

List Functions

- **Reverse, Length and Extend**

```
my_list.reverse()
print(my_list)
print(len(my_list))



fruits = ["apple", "banana"]
fruits.append("cherry")
print(fruits)  # Output: ["apple", "banana", "cherry"]
vegetables = ["potato", "tomato"]
fruits.extend(vegetables)
print(fruits)  # Output: ["apple", "orange", "banana", "cherry", "potato", "tomato"]
```

List Functions

- **Min, max, in, copy**

```
new_fruits = fruits.copy()
new_fruits.append("mango")
print(fruits)  # Output: ["apple", "banana", "cherry"] (original list is not modified)
print(new_fruits)  # Output: ["apple", "banana", "cherry", "mango"]
is_mango_in_fruits = "mango" in fruits
print(is_mango_in_fruits)  # Output: False
numbers = [5, 1, 8, 2]
smallest = min(numbers)
print(smallest)  # Output: 1
numbers = [5, 1, 8, 2]
largest = max(numbers)
print(largest)  # Output: 8
```

List Access

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(magician)
```

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
```

Numerical List

```
for value in range(1, 5):
    print(value)


print("Append to list")
list = []
for value in range(1, 5):
    list.append(value)
print(list)
```

List

- Looping through a slice or sublist

```
players = ['charles', 'martina', 'michael', 'florence', 'eli', 'atanu', 'jacob']
print("Here are the first three players on my team:")

for player in players[2:5]:
    print(player.title())
```

Try Out

- Python program to find unique numbers in a given list.
- Python program to create a list of 5 random integers.
- Write a Python program that takes a list of numbers as input and returns a new list containing only the even numbers.
- Imagine, one list contains fruit names [‘banana’, ‘mango’] and another one contains number of fruits [10,25]. Arrange them in a new list such a way that [‘banana’, 10, ‘mango’, ‘25’]

Try Out

- Write a Python program that takes a list containing duplicate elements and returns a new list with the duplicates removed, sorted in ascending order.
- Write a Python program that takes a list and an element as input and returns the index of the first occurrence of the element in the list, or -1 if the element is not present.

Tuple

- In Python, a tuple is an ordered collection of items that are similar to lists. However, unlike lists, tuples are immutable, which means you cannot modify their elements after they are created.

```
my_tuple = ("apple", "banana", "cherry")
```

Tuple

- Ordered: Tuples maintain the order in which you add elements.
- Immutable: Once created, you cannot change the elements within a tuple.
- Heterogeneous: Tuples can store elements of different data types.
- Duplicates: Tuples allow duplicate elements within the collection.

Tuple

```
#Creation
my_tuple = ("apple", "banana", "cherry")
print(my_tuple)

# List to Tuple
my_list = ["apple", "banana", "cherry"]
my_tuple = tuple(my_list)
print(my_tuple)
```

Tuple

- Access

```
my_tuple = ("apple", "banana", "cherry")
first_fruit = my_tuple[0]  # first_fruit will be "apple"
print("Single Element",first_fruit)

for fruit in my_tuple:
  print(fruit)
```

Tuple

- Can’t modify

```
dimensions = (200, 50)
dimensions[0] = 250 #Error
```

Reassign is possible

```
dimensions = (400, 33)
```

Tuple Functions

- Min, Max, Length, Sorted, In, Concatenation

Dictionary

- Python dictionaries are another fundamental data structure used to store collections of data. Unlike lists that use numerical indexes, dictionaries use key-value pairs for accessing elements.

- Properties of Dictionaries:
  - Unordered: The order in which you add key-value pairs to a dictionary is not preserved.
  - Changeable: You can modify or remove elements after creating a dictionary.
  - Heterogeneous Keys: Keys can be of various data types (strings, numbers, tuples). Keys must be unique within a dictionary.
  - Values can be Duplicated: Values can have duplicate data within the dictionary.

Dictionary

```
#Declare
alien_0 = {'color': 'green', 'points': 5}
print(alien_0['color'])
print(alien_0['points'])
my_dict = dict(name="Bob", age=25)
print(my_dict)
```

Dictionary

#Access

```
my_dict = {"name": "Alice", "age": 30, "city": "New York"}
name = my_dict["name"]  # name will be "Alice"
```

#Add or modify

```
my_dict = {"name": "Alice", "age": 30}
my_dict["city"] = "New York"  # Adding a new key-value pair
my_dict["age"] = 31  # Modifying an existing value
print(my_dict)
```

Dictionary Usage

```
if my_dict["age"]<31:
    print("Not too old")
elif my_dict["age"] == 31:
    print("About to be to buira")
else:
    print("Definitely Buro manush")
```

Dictionary

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'rust',
    'phil': 'php',
}
print(favorite_languages['jen'])
del favorite_languages['jen']
print(favorite_languages)
```

Dictionary

- Not found

```
var1 = favorite_languages.get("sarah")
print(var1)
var1 = favorite_languages.get("atanu", "not found")
print(var1)
```

Dictionary

- Looping through data

```
for k, v in favorite_languages.items():
    stri = f"Key {k} and value is {v}"
    print(stri)

#Only keys
for k in favorite_languages.keys():
    print(k)
```

Dictionary

Sorted Access

```
for k in sorted(favorite_languages.keys()):
    print(k)
```

List in dictionary?

```
pizza = {
    'crust': 'thick',
    'toppings': ['mushrooms', 'extra cheese'],
}
for k, v in pizza.items():
    print(f"Key is {k} and value is {v}")
```

Try Out

- Write a Python program that takes a dictionary as input and returns a new dictionary where the keys and values are swapped. If there are duplicate values, the new key should be the first occurrence of the value.

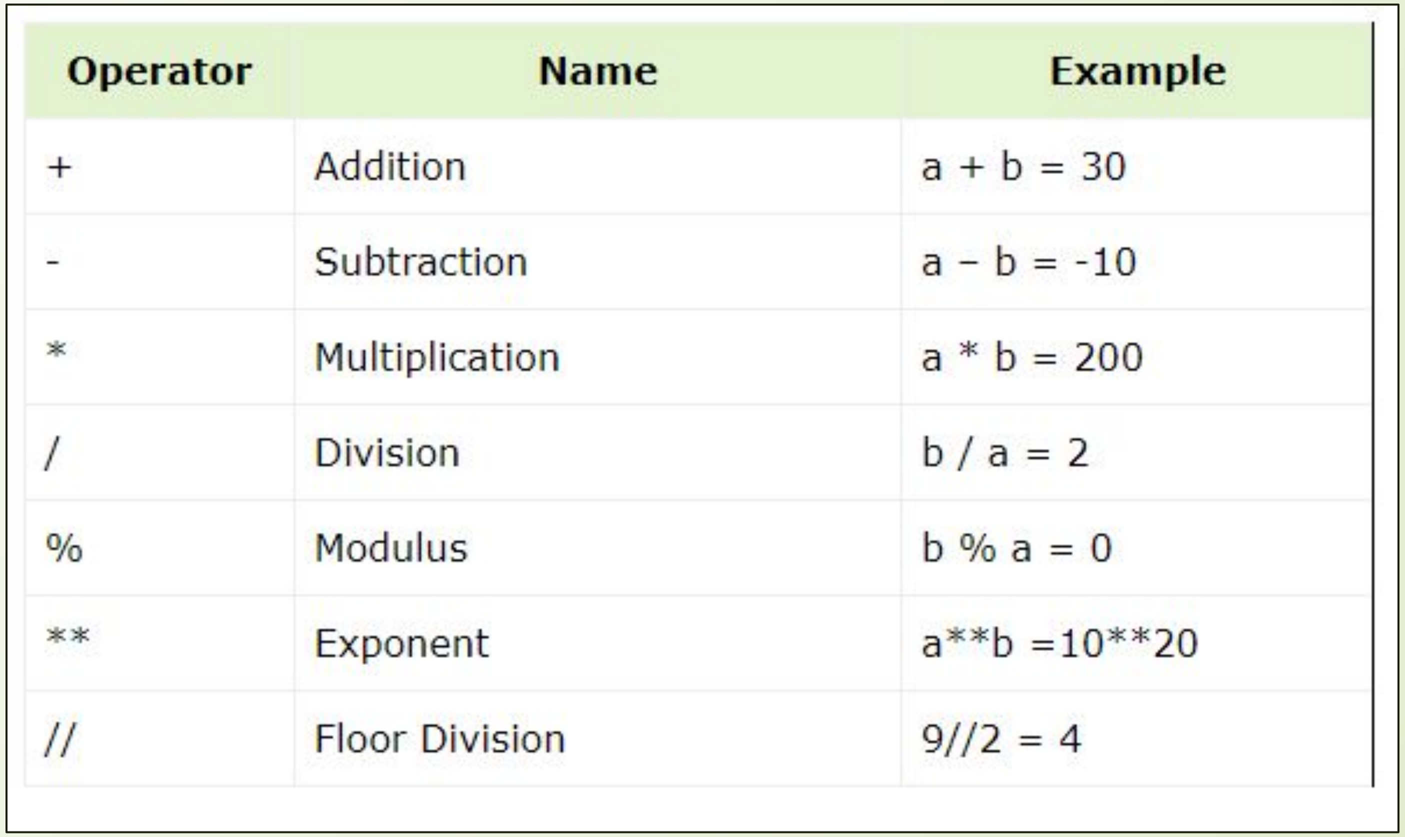
```
input_dict = {"name": "Alice", "age": 30, "city": "New York"}

# Potential Output: {30: 'age', 'New York': 'city', 'Alice': 'name'}
```

Try Out

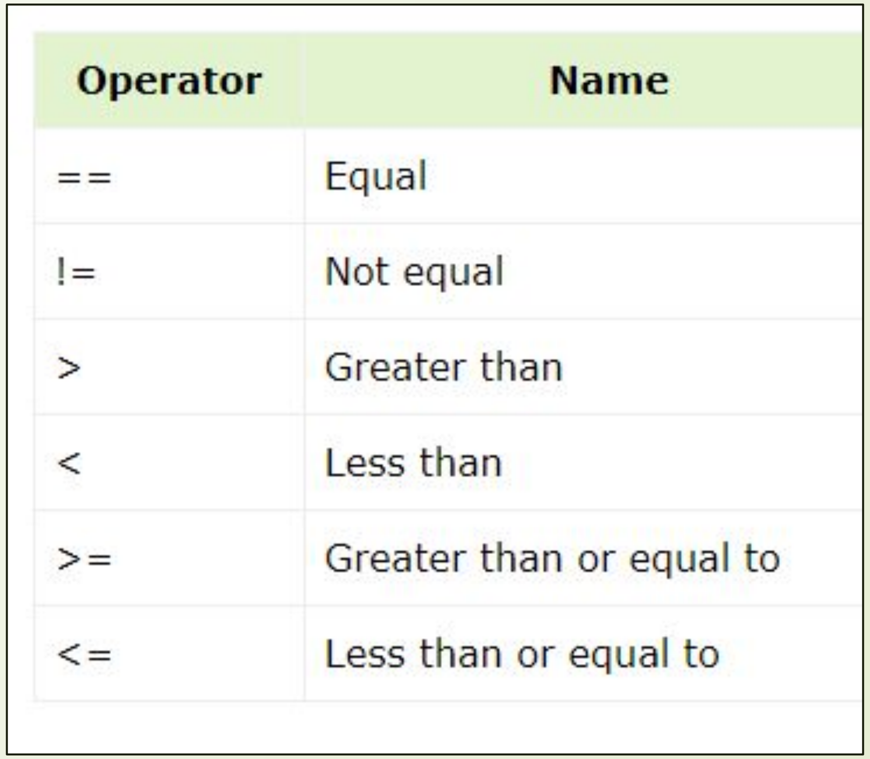
- Create a phonebook application using dictionaries. The dictionary can store contact information where keys are names (strings) and values are dictionaries containing details like phone number (string) and email address (string, optional). Write functions to:
  - Add a new contact to the phonebook.
  - Search for a contact by name and display their details.
  - Update the contact information for an existing contact.
  - Delete a contact from the phonebook.

Python Operators (Arithmatic)



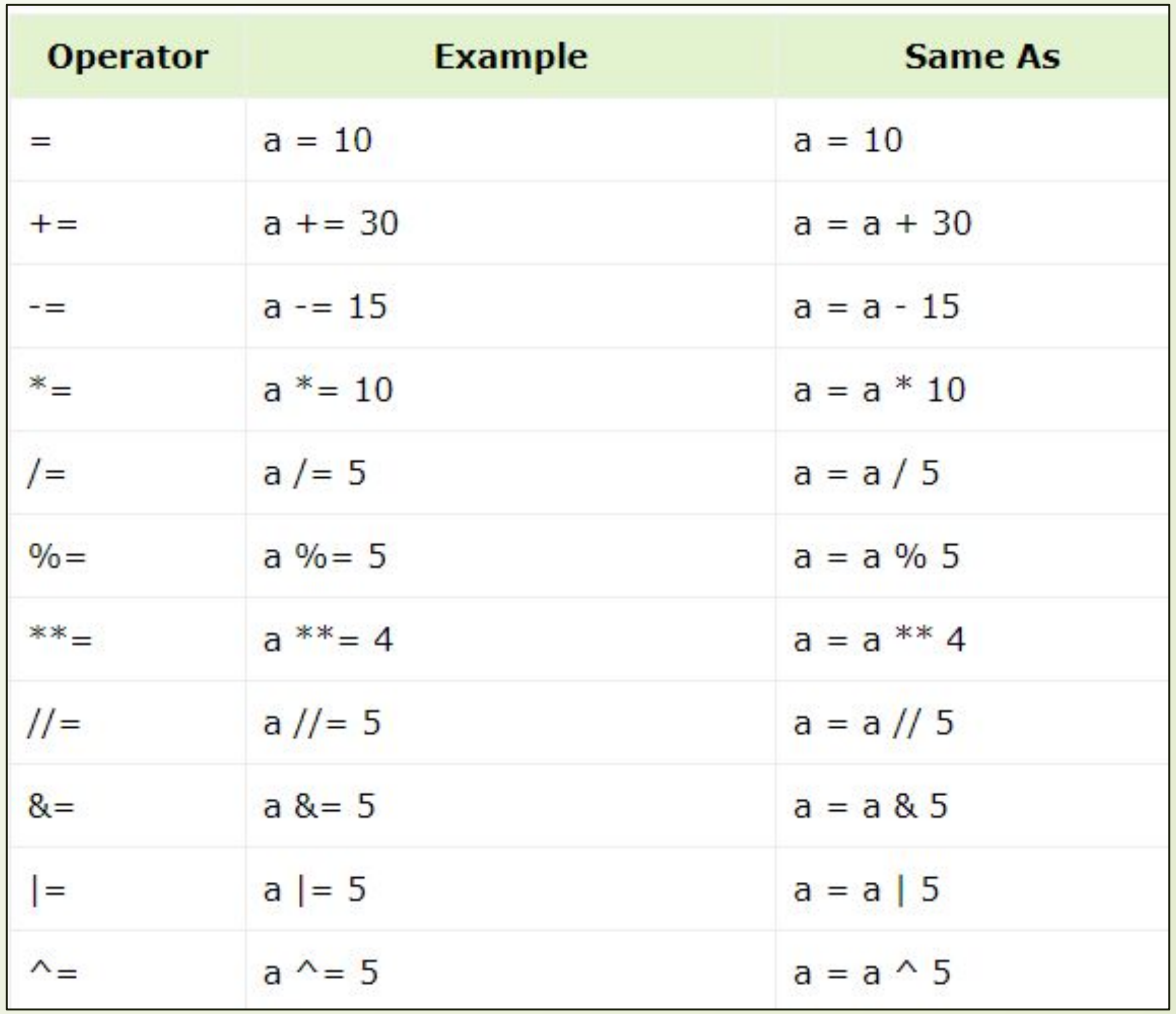
Operator	Name	Example
+	Addition	a + b = 30
-	Subtraction	a – b = -10
*	Multiplication	a * b = 200
/	Division	b / a = 2
%	Modulus	b % a = 0
**	Exponent	a**b =10**20
//	Floor Division	9//2 = 4

Python Operators (Comparison)



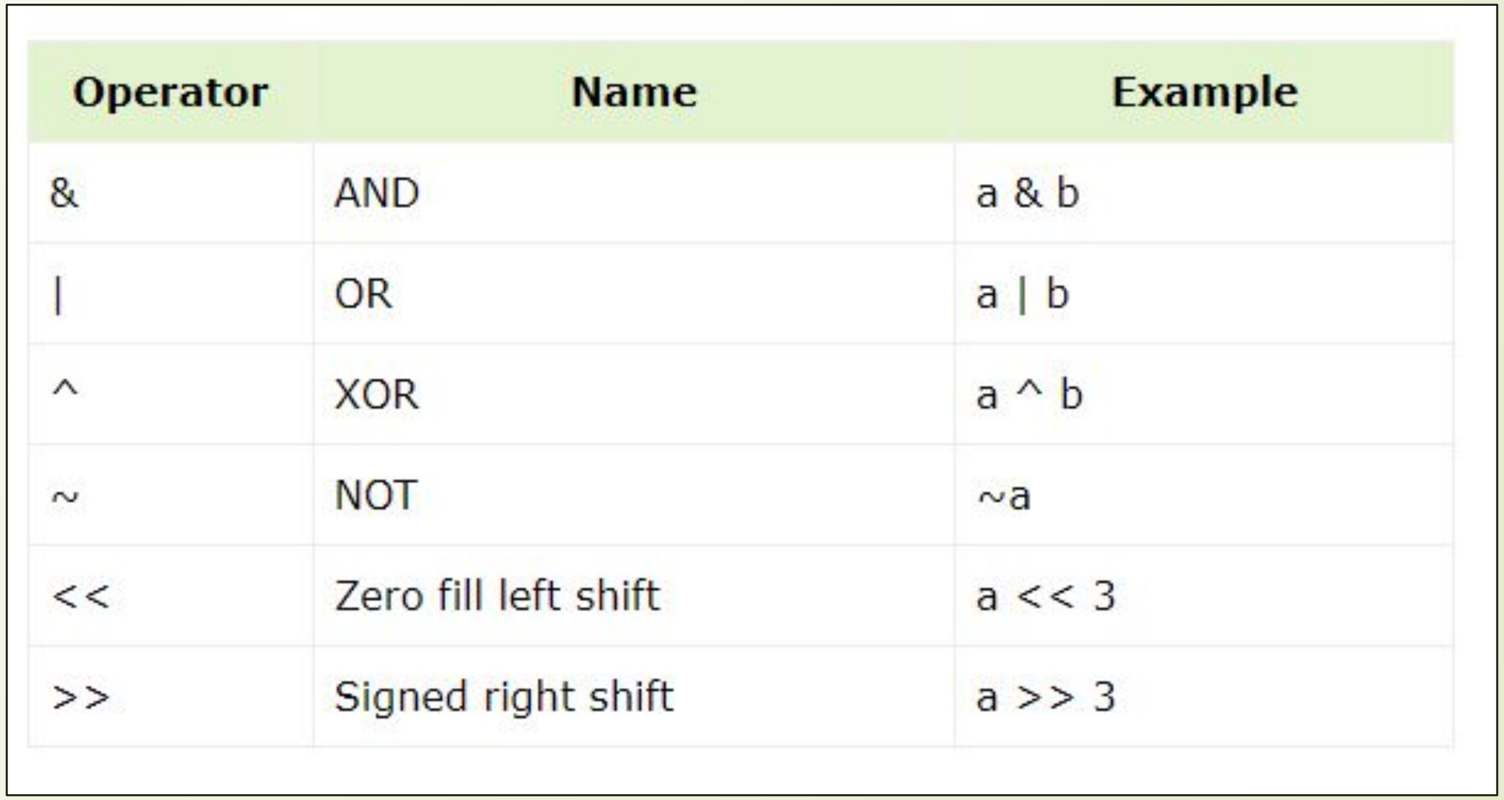
Operator	Name
==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Python Operators (Assignment)



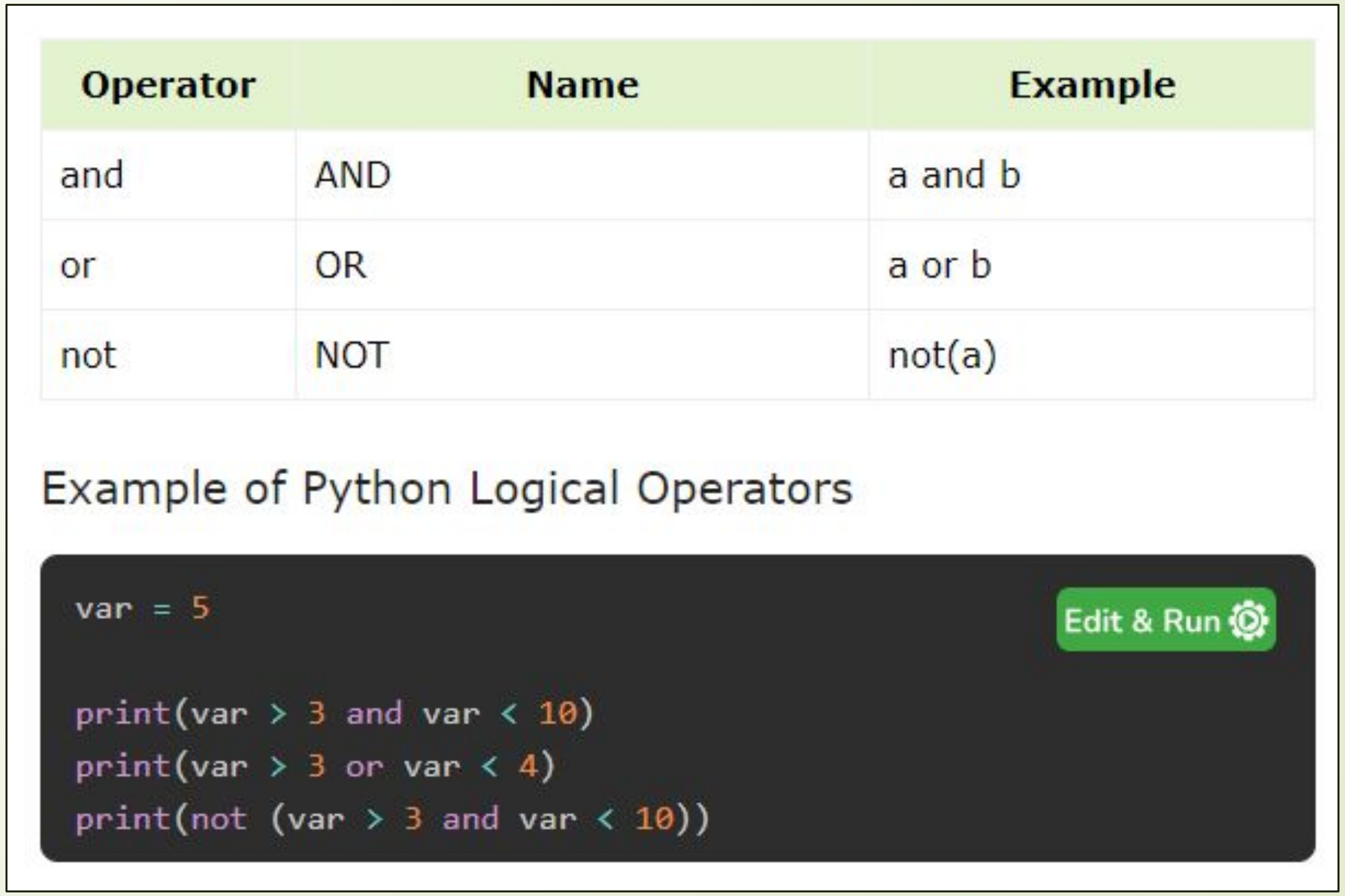
Operator	Example	Same As
=	a = 10	a = 10
+=	a += 30	a = a + 30
-=	a -= 15	a = a - 15
*=	a *= 10	a = a * 10
/=	a /= 5	a = a / 5
%=	a %= 5	a = a % 5
**=	a **= 4	a = a ** 4
//=	a //= 5	a = a // 5
&=	a &= 5	a = a & 5
|=	a |= 5	a = a | 5
^=	a ^= 5	a = a ^ 5

Python Operators (Bitwise)



Operator	Name	Example
&	AND	a & b
|	OR	a | b
^	XOR	a ^ b
~	NOT	~a
<<	Zero fill left shift	a << 3
>>	Signed right shift	a >> 3

Python Operators (Logical)



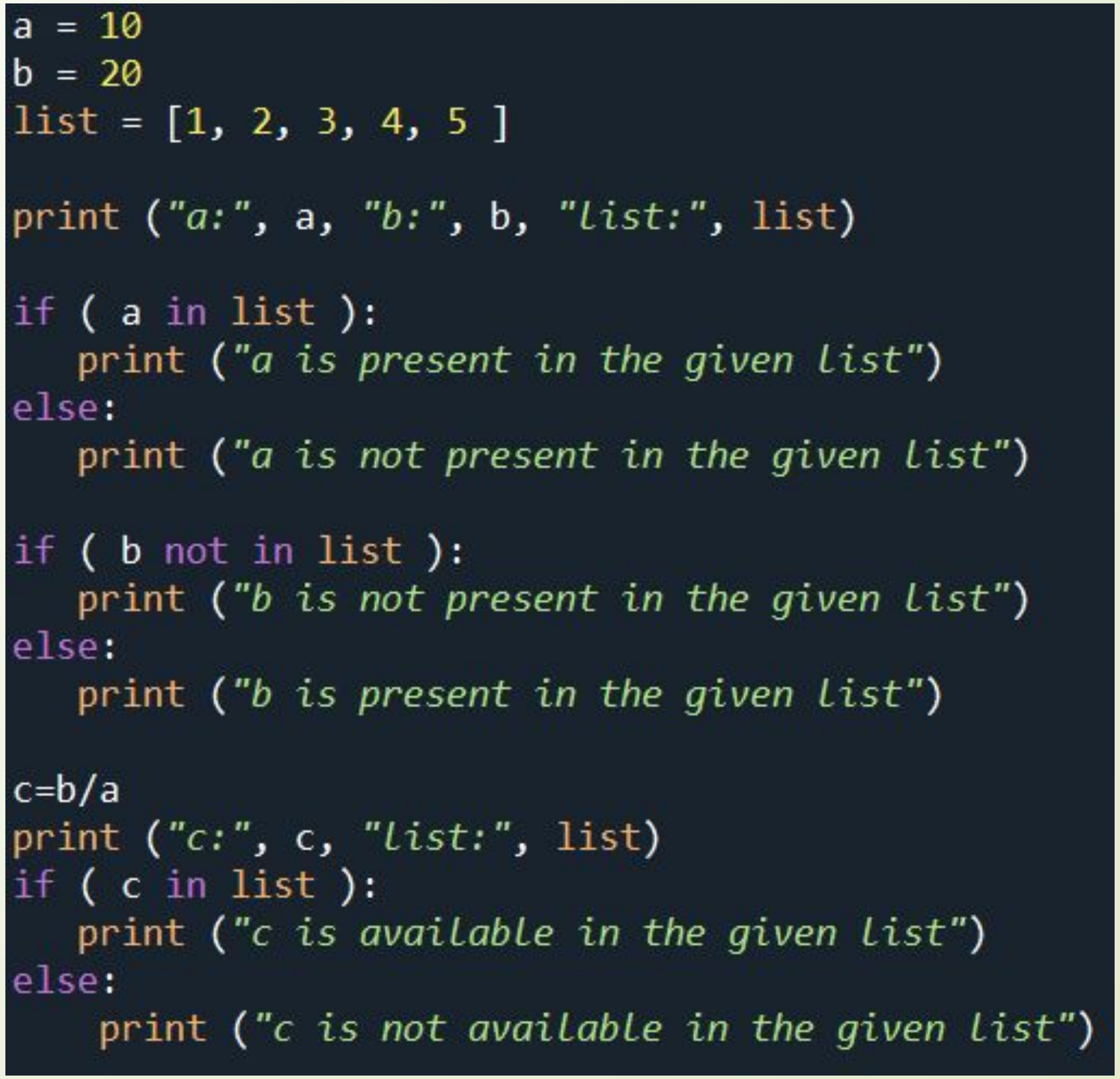
Operator	Name	Example
and	AND	a and b
or	OR	a or b
not	NOT	not(a)

Example of Python Logical Operators

```
var = 5

print(var > 3 and var < 10)
print(var > 3 or var < 4)
print(not (var > 3 and var < 10))
```

Python Operators (In and Not-in)



```
a = 10
b = 20
list = [1, 2, 3, 4, 5 ]

print ("a:", a, "b:", b, "list:", list)

if ( a in list ):
   print ("a is present in the given list")
else:
   print ("a is not present in the given list")

if ( b not in list ):
   print ("b is not present in the given list")
else:
   print ("b is present in the given list")

c=b/a
print ("c:", c, "list:", list)
if ( c in list ):
   print ("c is available in the given list")
else:
    print ("c is not available in the given list")
```